

Spring 3-1-1974

BRNANL, a Fortran Program to Identify Basic Blocks in Fortran Programs ; CU-CS-040-74

Lloyd D. Fosdick

University of Colorado Boulder

Follow this and additional works at: http://scholar.colorado.edu/csci_techreports

Recommended Citation

Fosdick, Lloyd D., "BRNANL, a Fortran Program to Identify Basic Blocks in Fortran Programs ; CU-CS-040-74" (1974). *Computer Science Technical Reports*. 39.

http://scholar.colorado.edu/csci_techreports/39

This Technical Report is brought to you for free and open access by Computer Science at CU Scholar. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.

BRNANL, A Fortran Program to
Identify Basic Blocks in Fortran Programs*

by

Lloyd D. Fosdick
Department of Computer Science
University of Colorado
Boulder, Colorado

Report #CU-CS-040-74

March, 1974

* This work supported in part by National Science Foundation Grant GJ-660
and National Science Foundation Grant GJ-36461

Abstract

A basic block is a sequence of consecutive Fortran statements which must be executed consecutively; that is, if one statement in the block is executed, all are executed. Except for special cases noted in the text, a Fortran program is a catenation of basic blocks. BRNANL is a Fortran program designed to recognize basic blocks in a Fortran program. Given a Fortran program (FP) BRNANL will generate a modified Fortran program (MFP) in which a subroutine call is located at the head of every basic block. Execution of the MFP produces the same results as execution of the FP but the inserted subroutine calls permit monitoring of the execution sequences. User information for running BRNANL is presented.

Keywords: Software testing, control path analysis.

1. Introduction

This report describes external features of a program BRNANL which is designed to identify basic blocks in an ANSI Fortran [1] program. Informally, a basic block (BB) is a sequence of statements which must be executed consecutively: a precise definition will be given later. The following example serves to illustrate the idea of a BB in Fortran:

```
      .  
      .  
      .  
      K = K + 1  
      IF(K) 10, 20, 30 } tail end of BB  
  
10 X = X + Y } BB  
  
20 Y = 3.0*Z + 9.0  
   D = A*B + C*D  
   GO TO(40, 50), J } BB  
  
40 A = 5.0  
   .  
   .  
   . } head end of BB
```

The notion of a basic block which we use is similar, but not identical, to that used in the specification of ANSI Fortran ([1], section 10.2.7). It follows more closely the definition generally used in the code optimization literature [2, 3, 4, 5]. Programs similar to this have appeared before. The program which is most similar to ours is one called FETE [6] written by Ingalls*, another program of this type appears to have been contained in a larger program reported by Allen [4]. A slightly different, but related program has been reported by Russell and Estrin [7]. We desired a program of this type, written in ANSI Fortran for portability, which could be easily

* An improved version, called FORTUNE, is commercially available from Capex (Phoenix, Arizona).

modified to meet various needs we had in connection with a project on software validation. For these reasons we created the program described here.

BRNANL accepts as input a syntactically correct ANSI Fortran program, say FP, and produces a modified form of FP, say MFP, differing from FP in that a subroutine call has been placed at the beginning of every BB. BRNANL numbers the BBs in order of their appearance in the source code and this number appears as a parameter in the inserted subroutine call. There is a second calling parameter which is used to identify special situations. A subroutine call is inserted before the first executable statement of a program; the second parameter has the value 1 in this case. A subroutine call is inserted before every STOP statement of a program; the second parameter has the value 3 in this case. In the normal case, when the inserted call appears as the first statement of a BB the second parameter has the value 2. The MFP for the example above is shown below.

```
      .  
      .  
      .  
      K = K + 1  
      IF(K) 10, 20, 30  
10    CALL XXXXXX(5, 2)  
      X = X + Y  
20    CALL XXXXXX(6, 2)  
      Y = 3.0*Z + 9.0  
      D = A*B + C*D  
      GO TO (40, 50), J  
40    CALL XXXXXX(7, 2)  
      A = 5.0  
      .  
      .  
      .
```

Here the first BB of the segment has been arbitrarily numbered 5 and the called subroutine arbitrarily named XXXXXX.

The MFP produced by BRNANL has the physical form of a printed listing and/or a punched deck and/or a file which may be on disc or tape depending on the system under which it executes. All statements inserted by BRNANL in the MFP are flagged by asterisks in columns 73-80 of the output. The printed listing and the file have the block number of each statement and the sequential line number recorded in columns to the right of each statement. A copy of the FP and the MFP are shown in Appendix E for a subroutine subprogram.

It is possible to suppress entirely the insertion of the subroutine calls. This is controlled by a datum on a data card read by BRNANL (cf. Appendix A). When this option is used, the listing which is produced will have the block number and line number at the right of each statement as before. This option is used to analyze the structure of the flowgraph for the program. In this report we are primarily concerned with using BRNANL to obtain a MFP which does have the subroutine calls inserted, so no further consideration is given to this option.

When the MFP is executed, various types of information can be recorded depending on the subroutine XXXXXX*. For example, the set of basic blocks executed can be recorded, the frequency of execution of basic blocks can be recorded, the sequence in which basic blocks are executed can be recorded, etc. Used in this way BRNANL is a valuable tool in program testing and it

* Henceforth we will use XXXXXX for the name of the subroutine appearing in the inserted call.

was with this purpose in mind that BRNANL was constructed. Since its construction, we have found it to be a useful tool in the reduction of a Fortran program to a directed graph.

Although BRNANL was written in ANSI Fortran it does contain one machine dependent subroutine CHRCHK which is designed to classify a character which has been read with an A1 format specification as a letter, a digit, or a special character. Specifications of this subroutine will be found in Appendix D.

Use of BRNANL is very simple. One card containing parameter specifications is placed in front of the FP and one card containing the character \$ in column 7 is placed in back of the FP: the resulting deck is the data deck for BRNANL. This deck setup is shown in Appendix B. The first card contains the following parameter specifications: name of the subroutine for the inserted call; a unique Fortran variable name (i.e. a name not used in the FP); initial block number; initial line number; flag to indicate suppression of inserted subroutine calls. Details are in Appendix A. The total storage required for the assembled program on the CDC 6400 computer operating under KRONOS 2.1 using the RUN compiler is 4240₈ words. In this environment sixty-six seconds of central processor time was required to run BRNANL on itself which consists of 1737 source statements excluding comments.

2. Basic Blocks

In this section the precise rules for identifying basic blocks and inserting subroutine calls are given. As already indicated, a BB is a sequence of one or more consecutive, executable statements in a source program. Suppose we identify the consecutive executable statements in a source program as S_1, S_2, \dots, S_n . Each BB consists of some subsequence, say $S_j, S_{j+1}, \dots, S_{j+k}$; the first statement, S_j is called the head, the last statement, S_{j+k} , is called the tail, and the sequence of statements between the head and the tail, $S_{j+1}, S_{j+2}, \dots, S_{j+k-1}$ is called the trunk. The trunk may be empty and the head and tail may be embodied in a single statement.

A tail is any one of the following:

- (a) logical IF statement;
- (b) arithmetic IF statement;
- (c) DO statement;
- (d) any form of GO TO statement;
- (e) RETURN statement
- (f) STOP statement;
- (g) any statement followed by a labelled statement except when the labelled statement is a FORMAT statement or when the labelled statement is the terminal statement in a DO loop;
- (h) the terminal statement in a DO loop.

A head is the statement immediately following a tail with two exceptions: the terminal statement in a DO-loop is never a head; the first executable statement of the main program and every subprogram is a head.

From this definition, excepting three special situations discussed below, the following assertions are true:

- (a) Every S_i belongs to a BB and cannot belong to more than one BB.
- (b) If any S_i in a BB is executed then every statement in that BB is executed.

From the first assertion it follows that we may view a program as a catenation of basic blocks. From the second assertion we may conclude that if the head statement of every BB of the program is executed then every statement of the program is executed.

3. Exceptional Situations

The logical IF statement presents one exceptional situation. Consider the statement:

```
IF(K.LT.0) X = X + Y.
```

This statement is a tail, however execution of this tail does not necessarily imply execution of the embedded assignment statement $X = X + Y$. To resolve this we treat only the structure

```
IF(<logical expression>)
```

as a tail and the portion of the logical IF following this is treated as a BB consisting of one statement. Thus in the above example IF(K.LT.0) would be the tail for, say, BB(12), then the statement $X = X + Y$ would be BB(13). With this understanding the two assertions above remain valid.

The second exceptional situation arises when a jump within a DO-loop can go to the last statement in the scope of a DO, as illustrated in the following situation

```
DO 20 J = 1, K
  X(J) = X(J) + Y
  IF(X(J)) 10, 20, 30
10  L = L + 1
20  V(J) = 0
30  A = B + C
```

Since there is a jump possible to statement

```
20  V(J) = 0
```

we ought to identify it as a head -- it is a tail by virtue of it being the terminal statement in a DO-loop (rule (k) above). However, if we were to treat this statement also as a head, then we would have in the MFP

```
20  CALL XXXXXX(--,--)
    V(J) = 0
```

violating the DO-loop. We resolve this problem by not permitting the terminal statement in a DO-loop to be a head; it is always treated only as a tail. The MFP for the program segment above is

```
      DO 20 J = 1, K
      CALL XXXXXX(12, 2)
      X(J) = X(J) + Y
      IF(X(J)) 10, 20, 30
10    CALL XXXXXX(13, 2)
      L = L + 1
20    V(J) = 0
      CALL XXXXXX(14, 2)
30    CALL XXXXXX(15, 2)
      A = B + c
```

where the BB numbering arbitrarily starts at 12 and BB(14) is a dummy used to detect satisfying the DO-loop. Thus the pair of statements

```
10    L = L + 1
20    V(J) = 0
```

is BB(13) and it is evident that assertion (b) above is not true. On the other hand it is important to note that if the head of every BB (including the dummy) is executed, then it is true that every statement has been executed. Also the number of times the statement

```
20    V(J) = 0
```

is executed is given by the expression

$$n_{12} - n_{11} + n_{14}$$

where n_i is the number of times BB(i) is entered. Finally, we observe that if DO-loops are terminated with CONTINUE statements, a good programming practice in any case, then jumps to the end of a DO-loop do not cause any important difficulty since one is not usually interested in the execution of CONTINUE statements.

The third exceptional situation arises when there is no return to the calling program after a subprogram has been called into execution. For example, in the basic block,

```

    --
    --
    --
30   J = J + 1
      X = X + Y
      CALL XAMPL(X, J, Z)
      X = Z
      GO TO 20
    --
    --
    --

```

failure to return from XAMPL makes assertion (b) above false. If STOP statements are permitted only in the main program, then this situation cannot arise unless execution is aborted by the system due to an error (overflow, array bounds violation, etc.).

These special situations could be eliminated. The logical IF problem could be removed by a replacement of the logical IF by an arithmetic IF and suitable restructuring of the program. The DO-loop problem could be removed by permitting assignment of new statement labels and appropriate re-labelling. Finally, a change in the rules defining a BB could partially eliminate the STOP statement problem. A subroutine CALL statement could be a BB but there would still be a problem with FUNCTION calls since these are embedded in statements.

The splitting of a logical IF statement in two BBs is done in the following way. Suppose we have the logical IF statement

<label> IF(<Boolean expression>)<statement>

then in the MFP this appears as

```
<label> LLLLLL = <Boolean expression>  
        IF(LLLLLL) CALL XXXXXX(--,--)  
        IF(LLLLLL)<statement>
```

It is evident that the subroutine call will be executed if and only if <statement> is executed so the call can be associated with the BB for the statement. The name LLLLLL is arbitrary: it is the second parameter on the data card. The MFP will also have a type declaration:

```
LOGICAL LLLLLL
```

Since a logical IF can terminate a DO-loop, it is evident that this situation presents a special problem. Splitting of the logical IF is not done in this case. Thus in the following sequence

```
DO 10 I = 1, N  
  --  
  --  
  --  
10  IF(X.LT.0) X = 1
```

The "BB" X = 1 is not identified as a BB. If the assertion (b) is to be valid, it is evident that a logical IF terminating a DO-loop must be prohibited. When BRNANL detects this situation it prints an error message.

In our own use of BRNANL we preprocess the FP with another program, STYLE [8], which reformats the FP and causes each DO loop to terminate on a CONTINUE statement. This essentially removes the difficulties cited above.

4. First Executable Statement and STOP Statement.

The first BB executed in a program is given special treatment. Suppose the FP begins with the statements

```
C THIS IS THE MAIN PROGRAM
  DIMENSION A(10), B(10, 10)
10  READ (5, 999) A
    DO 20 I = 1, 10
      ---
      ---
      ---
```

Then the MFP begins with the statements

```
C THIS IS THE MAIN PROGRAM
  DIMENSION A(10), B(10, 10)
    CALL XXXXXX(0, 1)
10  CALL XXXXXX(1, 2)
    READ(5, 999) A
    DO 20 I = 1, 10
```

The second parameter of the first CALL is 1, uniquely identifying it as preceding the first executable statement in the program. This information allows the routine XXXXXX to perform initialization. The first parameter in this call is one less than the initial block number, the third parameter on the data card (cf. Appendix A); here it is assumed that this number was 1. The second CALL, which would be there even if the label were not on the READ statement, is used to identify actual entry into the BB. It is to be noted that this situation arises only in a main program.

The BB in which a STOP appears as the tail is treated in a special way. In addition to the call which is inserted at the head of the block, a call is inserted immediately before the STOP statement. The following example illustrates this. Suppose the FP contains the BB, say BB(100),


```
150  X = SIN(Y)
      WRITE(6, 999) X, Y
      STOP
```

Then the MFP is

```
150  CALL XXXXXX(100, 2)
      X = SIN(Y)
      WRITE(6, 999) X, Y
      CALL XXXXXX(100, 3)
      STOP
```

It is to be noted that the second parameter in the CALL just before the STOP is 3; this special value is used only before a STOP so the routine XXXXXX can take whatever steps are appropriate for such a condition.

Typically it would print accumulated data on BB activity in executing the MFP.

5. Limitations.

The most important limitation arises from the fact that BRNANL assumes that the FP is a syntactically correct ANSI Fortran program; if it is not, incorrect execution may result.

Other limitations are listed below:

1. Maximum number that can be assigned to a BB is 9999;
2. Maximum number of subscripted variables in each program unit (subroutine subprogram, function subprogram, main program) is 50;
3. Maximum depth for DO-loop nesting is 29.

It is recommended that all DO-loops terminate on CONTINUE statements. (Preprocessing the FP by STYLE [8] will guarantee this.) If a DO-loop does not terminate on a CONTINUE statement BRNANL will still execute properly, however jumps to the last statement in the DO need special consideration as described in section 3 of this report.

A list of error messages which can be produced by BRNANL is given in Appendix B.

6. I/O FILES.

BRNANL reads the input file from unit 5, writes the print file on unit 6, and writes the punch file on unit 7. Specifically all READ statements have the form

READ(KIN, ...

all WRITE statements for producing the listing of the MFP and any error messages have the form

WRITE(KPR, ...

and all write statements for producing the source "deck" for the MFP have the form

WRITE(KPU, ...

A DATA statement is used assign 5, 6, 7 to KIN, KPR, KPU, respectively.

7. Acknowledgements.

Much of the testing, corrections to errors in the original program, and insertion of COMMENTS was done by Jeffery Wright and Jacob Wu, graduate students in the Department of Computer Science at the University of Colorado.

8. References.

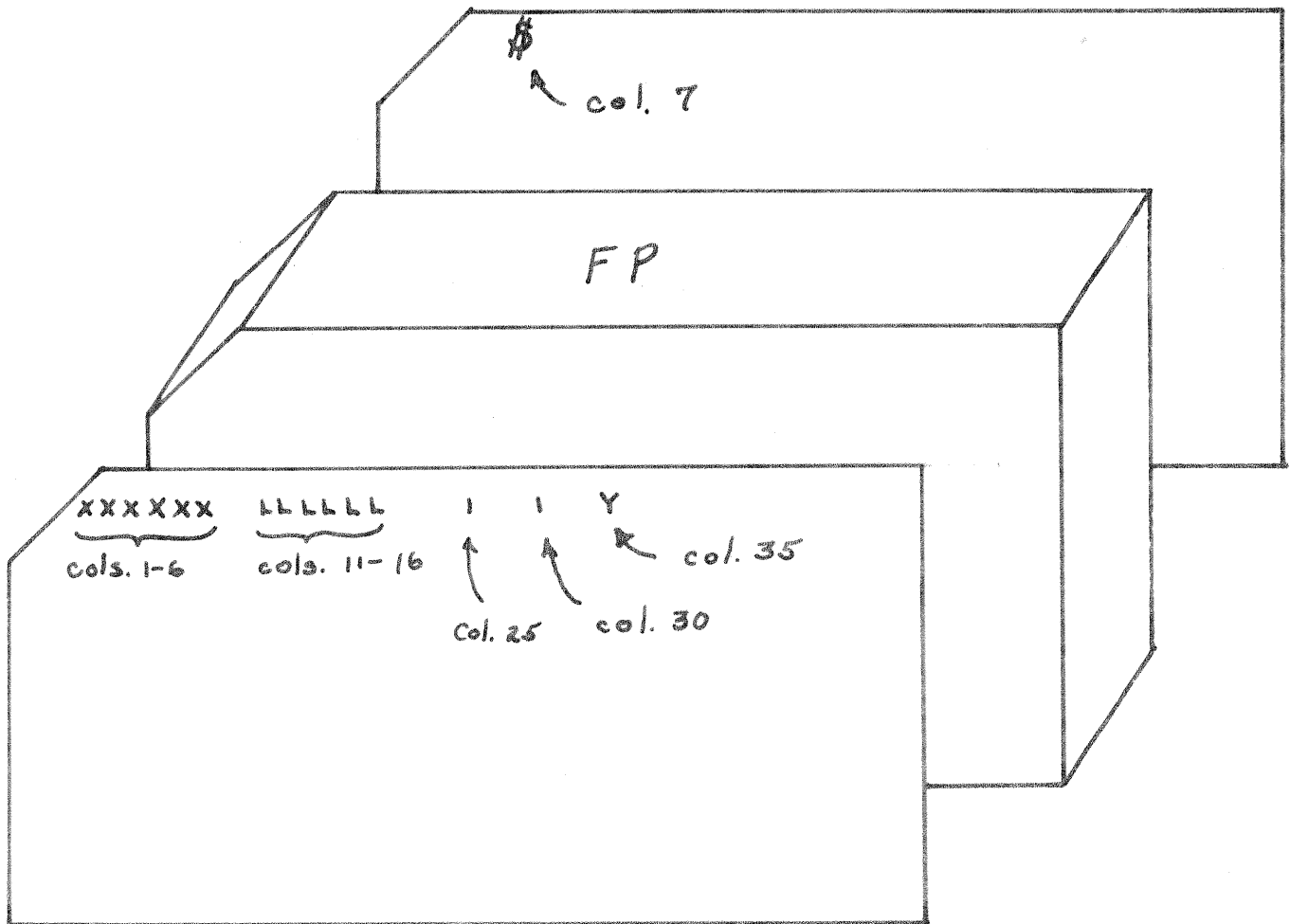
1. American National Standard Fortran, American National Standards Inc. (1966).
See also:
 - (a) Fortran vs. Basic Fortran, Comm ACM 7 (Oct. 1964), 591-625.
 - (b) Clarification of Fortran Standards - Initial Progress, Comm A M 12 (May 1969), 289-294.
 - (c) Clarification of Fortran Standards - Second Report, Comm ACM 14 (Oct 1971), 628-642.
2. J. Cocke and J. T. Schwartz, Programming Languages and Their Compilers, Courant Institute, NYU (1970).
3. F. E. Allen, Program optimization, in Annual Review in Automatic Programming, Vol. 5, Pergamon (1969).
4. F. E. Allen, Control flow analysis. ACM SIGPLAN Notices 5, 7 (1970), 1-19.
5. A. Aho and J. D. Ullman, The Theory of Parsing, Translation and Compiling, Volume II: Compiling: Prentice-Hall (1973).
6. D. H. H. Ingalls, FETE, A Fortran execution time estimator, Department of Computer Science, Stanford Univeristy, Report 71-204 (1971).
7. E. C. Russell and G. Estrin, Measurement based automatic analysis of Fortran programs. SJCC (1969), 723-732.
8. Dorothy Lang Wedel, STYLE Editor: User's Guide, Department of Computer Science, University of Colorado, Report 7 (1972).

Appendix A: Data card.

One data card must precede the FP. The layout of this card follows:

<u>cols.</u>	<u>contents</u>
1-6	Name of subroutine appearing in the inserted CALL statements;
11-16	Name of variable used to hold value of Boolean expressions appearing in logical IF statements;
22-25	Number of first BB in program; BBs are numbered sequentially in order of appearance in the FP; value entered as a right- justified integer;
26-30	Number of first line for sequential numbering of lines in output file supplied by BRNANL; value entered as a right- justified integer.
35	Y if subroutine calls are to be inserted.

Appendix B: Run Deck Organization



Appendix C: Error Messages

1: MORE THAN 20 CARDS USED FOR STATEMENT.

The maximum number of cards permitted for a statement is 20 (i.e. 19 continuation cards). Fatal error.

2: IF STATEMENT SYNTAX ERROR.

ANSI Fortran syntax error. Fatal error.

3: MORE THAN 50 DIMENSIONED VARIABLES.

An array in BRNANL called ARRNAM holds the list of dimensioned variables in the subprogram, or main program being processed. It is dimensioned at 50. Fatal error.

4: STATEMENT ENDS WITH LEFT PARENTHESIS.

ANSI Fortran syntax error. Fatal error.

5: DECLARATION FOLLOWED ONLY BY BLANKS.

ANSI Fortran syntax error. Fatal error.

6: LIMIT OF 9999 ON BASIC BLOCK INDEX EXCEEDED.

BRNANL requires BB index to lie in the interval (1, 9999). Fatal error.

7: NON-ANSI BLANK CARD ENCOUNTERED.

Blank cards are not permitted in the subject program. This card is automatically replaced by a blank comment card. Non-fatal error.

8: NON-ANSI PROGRAM CARD ENCOUNTERED.

A PROGRAM card is required for the main program in CDC Fortran, however this is not legal ANSI Fortran. This card is ignored. Non-fatal error.

9: LABEL CONTAINS AN ILLEGAL CHARACTER.

A label must consist of digits only. Fatal error.

10: LOGICAL IF CLOSING A DO-LOOP.

After each error message the following information is printed:

BUFFER A CONTAINS

---(statement being processed)

BUFFER B CONTAINS

---(next statement to be processed)

BUFFER D CONTAINS

---(first card of next statement)

Appendix D: Machine Dependent Subroutine

The subroutine CHRCHK in BRNANL is machine dependent and may have to be modified by the user for systems other than the CDC 6400. This subroutine determines whether a character is a letter, a digit, or special.

The subroutine specification is

SUBROUTINE CHRCHK(A, I, L)

where the formal parameters are defined as follows:

A -- a one dimensional array holding characters which have been read into it using an A1 format specification.

I -- the character to be checked is in position A(I).

L -- the routine CHRCHK makes the assignment

L = 1 if the character in A(I) is a letter

L = 2 if the character in A(I) is a digit

L = 3 if the character in A(I) is special.

A listing of this subroutine is on the following page.

```
      SUBROUTINE CHRCHK(A, I, L)
C THIS IS A CHARACTER CHECK ROUTINE FOR BUFFER A. L=1 IF
C A(I) IS A LETTER, L=2 IF A(I) IS A DIGIT, L=3 IF A(I) IS A
C SPECIAL CHARACTER.
      DIMENSION A(I)
      INTEGER ALPBL, ALPBH, NUML, NUMH, A
C THE FOLLOWING CHECKING TECHNIQUE SHOULD BE ADJUSTED FOR
C LOCAL CHARACTER SET IF ALPHABET CHARACTERS OR NUMERIC
C CHARACTERS ARE NOT
C IN A CONTIGUOUS GROUP.
      DATA ALPBL /1HA/, ALPBH /1HZ/, NUML /1H0/, NUMH /1H9/
      ICHR = A(I)
      IF (ICHR-ALPBL) 20, 10, 10
10    IF (ICHR-ALPBH) 40, 40, 20
20    IF (ICHR-NUML) 60, 30, 30
30    IF (ICHR-NUMH) 50, 50, 60
40    L = 1
      RETURN
50    L = 2
      RETURN
60    L = 3
      RETURN
      END
```

Appendix E: Example

On the following six pages an example illustrating the output obtained from BRNANL is shown. The first three pages contain the listing of the FP (a subroutine subprogram KZNONE). The next three pages contain the MFP as contained on the print file. In this example block numbering starts at 1 and line numbering start, at 10. Block numbers associated with logical IF statements are flagged by an asterisk.

```

SUBROUTINE KZEONE(X, Y, RE0, IM0, RE1, IM1)
C THE VARIABLES X AND Y ARE THE REAL AND IMAGINARY PARTS OF
C THE ARGUMENT OF THE FIRST TWO MODIFIED BESSEL FUNCTIONS
C OF THE SECOND KIND, K0 AND K1. RE0, IM0, RE1 AND IM1 GIVE
C THE REAL AND IMAGINARY PARTS OF  $\exp(X)*K0$  AND  $\exp(X)*K1$ ,
C RESPECTIVELY. ALTHOUGH THE REAL NOTATION USED IN THIS
C SUBROUTINE MAY SEEM INELEGANT WHEN COMPARED WITH THE
C COMPLEX NOTATION THAT FORTRAN ALLOWS, THIS VERSION RUNS
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX
C VARIABLES.

      DOUBLE PRECISION X, Y, X2, Y2, RE0, IM0, RE1, IM1,
      *R1, R2, T1, T2, P1, P2, RTERM, ITERM, EXSQ(8), TSQ(8)
      DATA TSQ(1) /0.000/, TSQ(2) /3.143036339206350-1/,
      *TSQ(3) /4.2907586229591500/, TSQ(4)
      * /2.9583744586966500/, TSQ(5) /5.4090315972444400/,
      *TSQ(6) /8.8040795780567600/, TSQ(7)
      * /1.3468535743251501/, TSQ(8) /2.0249916365870901/,
      *EXSQ(1) /0.564100308726400/, EXSQ(2)
      * /0.412026687498900/, EXSQ(3) /0.158488915795900/,
      *EXSQ(4) /0.30780033872550-1/, EXSQ(5)
      * /0.2778068429130-2/, EXSQ(6) /0.10000444123250-3/,
      *EXSQ(7) /0.10591155477110-5/, EXSQ(8)
      * /0.15224758042540-8/

C THE ARRAYS TSQ AND EXSQ CONTAIN THE SQUARE OF THE
C ABSCTISSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-
C HERMITE QUADRATURE.
      P2 = X*X + Y*Y
      IF (X.GT.0.000 .OR. R2.NE.0.000) GO TO 10
      WRITE (6,999999)
      RETURN

10 IF (R2.GE.1.9602) GO TO 50
   IF (R2.GE.1.84901) GO TO 30
C THIS SECTION CALCULATES THE FUNCTIONS USING THE SERIES
C EXPANSIONS
      X2 = X/2.000
      Y2 = Y/2.000
      P1 = X2*X2
      P2 = Y2*Y2
      T1 = -(DLG(P1+P2)/2.000+0.577215664901532900)
C THE CONSTANT IN THE PRECEDING STATEMENT IS EULER*S
C CONSTANT
      T2 = -DATAN2(Y,X)
      X2 = P1 - P2
      Y2 = X*Y2
      RTERM = 1.000
      ITERM = 0.000
      RE0 = T1
      IM0 = T2
      T1 = T1 + 0.500
      RE1 = T1
      IM1 = T2
      P2 = DSQRT(R2)
      L = 2.10600*P2 + 4.400
      IF (P2.LT.8.00-1) L = 2.12900*P2 + 4.000
      DO 20 N=1,L
        P1 = N
        P2 = N*N
        R1 = RTERM
        RTERM = (R1*X2-ITERM*Y2)/P2
        ITERM = (R1*Y2+ITERM*X2)/P2
        T1 = T1 + 0.500/P1
        RE0 = RE0 + T1*ITERM - T2*ITERM
        IM0 = IM0 + T1*ITERM + T2*ITERM
      20 CONTINUE

```

KZE 10
KZE 20
KZE 30
KZE 40
KZE 50
KZE 60
KZE 70
KZE 80
KZE 90
KZE 100
KZE 110
KZE 120
KZE 130
KZE 140
KZE 150
KZE 160
KZE 170
KZE 180
KZE 190
KZE 200
KZE 210
KZE 220
KZE 230
KZE 240
KZE 250
KZE 260
KZE 270
KZE 280
KZE 290
KZE 300
KZE 310
KZE 320
KZE 330
KZE 340
KZE 350
KZE 360
KZE 370
KZE 380
KZE 390
KZE 400
KZE 410
KZE 420
KZE 430
KZE 440
KZE 450
KZE 460
KZE 470
KZE 480
KZE 490
KZE 500
KZE 510
KZE 520
KZE 530
KZE 540
KZE 550
KZE 560
KZE 570
KZE 580
KZE 590
KZE 600
KZE 610
KZE 620
KZE 630

```

P1 = P1 + 1.000
T1 = T1 + 0.500/P1
RE1 = RE1 + (T1*ITERM - T2*ITERM)/P1
IM1 = IM1 + (T1*ITERM + T2*ITERM)/P1
20 CONTINUE
R1 = X/R2 - 0.500*(X*RE1 - Y*IM1)
R2 = -Y/R2 - 0.500*(X*IM1 + Y*RE1)
P1 = EXP(X)
RE0 = P1*RE0
IM0 = P1*IM0
RE1 = P1*RE1
IM1 = P1*IM1
RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL
C REPRESENTATION, EGN 3, EVALUATED WITH 15 POINT GAUSS-
C HERMITE QUADRATURE
30 X2 = 2.000*X
Y2 = 2.000*Y
R1 = Y2*Y2
P1 = USQRT(X2*X2+R1)
P2 = USQRT(P1*X2)
T1 = EXSQ(1)/(2.000*P1)
RE0 = T1*P2
IM0 = T1/P2
RE1 = 0.000
IM1 = 0.000
DO 40 N=2,8
T2 = X2 + TSQ(N)
P1 = USQRT(T2*T2+R1)
P2 = USQRT(P1+T2)
T1 = EXSQ(N)/P1
RE0 = RE0 + T1*P2
IM0 = IM0 + T1/P2
T1 = EXSQ(N)*TSQ(N)
RE1 = RE1 + T1*P2
IM1 = IM1 + T1/P2
40 CONTINUE
T2 = -Y2*IM0
RE1 = RE1/R2
R2 = Y2*IM1/R2
ITERM = 1.4142135623730900*DCOS(Y)
ITERM = -1.4142135623730900*SIN(Y)
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS OF COURSE,
C SQRT(2.0).
IM0 = RE0*ITERM + T2*ITERM
RE0 = RE0*ITERM - T2*ITERM
T1 = RE1*ITERM - R2*ITERM
T2 = RE1*ITERM + R2*ITERM
RE1 = T1*X + T2*Y
IM1 = -T1*Y + T2*X
RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE
C ASYMPTOTIC EXPANSIONS
50 ITERM = 1.000
ITERM = 0.000
RE0 = 1.000
IM0 = 0.000
RE1 = 1.000
IM1 = 0.000
P1 = 8.000*R2
P2 = USQRT(R2)
L = 3.9100+8.1201/P2
R1 = 1.000

```

KZE 640
KZE 650
KZE 660
KZE 670
KZE 680
KZE 690
KZE 700
KZE 710
KZE 720
KZE 730
KZE 740
KZE 750
KZE 760
KZE 770
KZE 780
KZE 790
KZE 800
KZE 810
KZE 820
KZE 830
KZE 840
KZE 850
KZE 860
KZE 870
KZE 880
KZE 890
KZE 900
KZE 910
KZE 920
KZE 930
KZE 940
KZE 950
KZE 960
KZE 970
KZE 980
KZE 990
KZE1000
KZE1010
KZE1020
KZE1030
KZE1040
KZE1050
KZE1060
KZE1070
KZE1080
KZE1090
KZE1100
KZE1110
KZE1120
KZE1130
KZE1140
KZE1150
KZE1160
KZE1170
KZE1180
KZE1190
KZE1200
KZE1210
KZE1220
KZE1230
KZE1240
KZE1250
KZE1260

KZE1270
KZE1280
KZE1290
KZE1300
KZE1310
KZE1320
KZE1330
KZE1340
KZE1350
KZE1360
KZE1370
KZE1380
KZE1390
KZE1400
KZE1410
KZE1420
KZE1430
KZE1440
KZE1450
KZE1460
KZE1470
KZE1480
KZE1490
KZE1500
KZE1510
KZE1520
KZE1530
KZE1540
KZE1550
KZE1560
KZE1570
KZE1580
KZE1590
KZE1600
KZE1610

```

P2 = 1.0D0
M = 8
K = 3
CO 60 N=1,L
M = M + 8
K = K - M
R1 = FLOAT(K-4)*R1
R2 = FLOAT(K)*R2
T1 = FLOAT(N)*P1
T2 = RTERM
RTERM = (T2*X+ITERM*Y)/T1
ITERM = (-T2*Y+ITERM*X)/T1
R0 = R0 + R1*ITERM
IM0 = IM0 + R1*ITERM
R1 = R1 + R2*ITERM
IM1 = IM1 + R2*ITERM
60 CONTINUE
T1 = DSQRT(P2*X)
T2 = -Y/T1
P1 = 8.86226925452758D-1/P2
C THIS CONSTANT IS SQRT(P1)/2.0, WITH P1=3.14159...
RTERM = P1*DCOS(Y)
ITERM = -P1*DSIN(Y)
R1 = R0*ITERM - IM0*ITERM
R2 = R0*ITERM + IM0*ITERM
R0 = T1*R1 - T2*R2
IM0 = T1*R2 + T2*R1
R1 = R1*ITERM - IM1*ITERM
R2 = R1*ITERM + IM1*ITERM
R1 = T1*R1 - T2*R2
IM1 = T1*R2 + T2*R1
RETURN
99999 FORMAT (42H ARGUMENT OF THE BESSEL FUNCTIONS IS ZERO,
* 35H OR LIES IN LEFT HALF COMPLEX PLANE)
END

```

```

SUBROUTINE KZEONE(X, Y, REO, IMO, REI, IMI)
LOGICAL LLLLLL
C THE VARIABLES X AND Y ARE THE REAL AND IMAGINARY PARTS OF
C THE ARGUMENT OF THE FIRST TWO MODIFIED BESSEL FUNCTIONS
C OF THE SECOND KIND, KO AND KI. REO, IMO, REI AND IMI GIVE
C THE REAL AND IMAGINARY PARTS OF EXP(X)*KO AND EXP(X)*KI,
C RESPECTIVELY. ALTHOUGH THE REAL NOTATION USED IN THIS
C SUBROUTINE MAY SEEM INELEGANT WHEN COMPARED WITH THE
C COMPLEX NOTATION THAT FORTRAN ALLOWS, THIS VERSION RUNS
C ABOUT 30 PERCENT FASTER THAN ONE WRITTEN USING COMPLEX
C VARIABLES.
      DOUBLE PRECISION X, Y, X2, Y2, REO, IMO, REI, IMI,
      * P1, R2, T1, T2, P1, P2, RTERM, ITERM, EXSQ(18), TSQ(18)
      DATA TSQ(1) /0.000/, TSQ(2) /3.19303633920635D-1/,
      * TSQ(3) /1.29075862295915D0/, TSQ(4)
      * /2.95837445869665D0/, TSQ(5) /5.40903159724444D0/,
      * TSQ(6) /8.80407957805676D0/, TSQ(7)
      * /1.34685357432515D1/, TSQ(8) /2.02499163658709D1/,
      * EXSQ(1) /0.564100308264D0/, EXSQ(2)
      * /0.4120286874989D0/, EXSQ(3) /0.1584889157959D0/,
      * EXSQ(4) /0.307800338255D-1/, EXSQ(5)
      * /0.2778068842913D-2/, EXSQ(6) /0.100004412325D-3/,
      * EXSQ(7) /0.105911554771D-5/, EXSQ(8)
      * /0.1522475804254D-8/
      C THE ARRAYS TSQ AND EXSQ CONTAIN THE SQUARE OF THE
      C ABSISSAS AND THE WEIGHT FACTORS USED IN THE GAUSS-
      C HERMITE QUADRATURE.
      CALL XXXXXX(0001,2)
      R2 = X*X + Y*Y
      LLLLLL=X.GT.0.00D0.UK.R2.NE.0.00D0
      IF(LLLLLL)CALL XXXXXX(0002,2)
      IF(LLLLLL)
      CALL XXXXXX(0003,2)
      *RITE (6,99999)
      RETURN
10 CALL XXXXXX(0004,2)
      LLLLLL=R2.GE.1.96D2
      IF(LLLLLL)CALL XXXXXX(0005,2)
      IF(LLLLLL)
      GO TO 50
      CALL XXXXXX(0006,2)
      LLLLLL=R2.GE.1.849D1
      IF(LLLLLL)CALL XXXXXX(0007,2)
      IF(LLLLLL)
      GO TO 30
      C THIS SECTION CALCULATES THE FUNCTIONS USING THE SERIES
      C EXPANSIONS
      CALL XXXXXX(0008,2)
      X2 = X/2.0D0
      Y2 = Y/2.0D0
      P1 = X2*X2
      P2 = Y2*Y2
      T1 = -(DLOG(P1+P2)/2.0D0+0.5772156649015329D0)
      C THE CONSTANT IN THE PRECEDING STATEMENT IS EULER'S
      C CONSTANT
      T2 = -DATAN2(Y,X)
      X2 = P1 - P2
      Y2 = X*Y2
      ITERM = 1.0D0
      REO = Y1
      IMO = T2
      T1 = T1 + 0.50D

```

KZE 10	0	10
*****	0	11
KZE 20	0	12
KZE 30	0	13
KZE 40	0	14
KZE 50	0	15
KZE 60	0	16
KZE 70	0	17
KZE 80	0	18
KZE 90	0	19
KZE 100	0	20
KZE 110	0	21
KZE 120	0	22
KZE 130	0	23
KZE 140	0	24
KZE 150	0	25
KZE 160	0	26
KZE 170	0	27
KZE 180	0	28
KZE 190	0	29
KZE 200	0	30
KZE 210	0	31
KZE 220	0	32
KZE 230	0	33
KZE 240	0	34
KZE 250	0	35
KZE 260	0	36
*****	1	37
KZE 270	1	38
*****	1	39
*****	2*	40
KZE 280	2*	41
*****	3	42
KZE 290	3	43
KZE 300	3	44
*****	4	45
*****	4	46
*****	5*	47
KZE 310	5*	48
*****	6	49
*****	6	50
*****	7*	51
KZE 320	7*	52
KZE 330	7	53
KZE 340	7	54
*****	8	55
KZE 350	8	56
KZE 360	8	57
KZE 370	8	58
KZE 380	8	59
KZE 390	8	60
KZE 400	8	61
KZE 410	8	62
KZE 420	8	63
KZE 430	8	64
KZE 440	8	65
KZE 450	8	66
KZE 460	8	67
KZE 470	8	68
KZE 480	8	69
KZE 490	8	70


```

RE1 = T1
IM1 = T2
P2 = DSQRT(R2)
L = 2.10600*P2 + 4.400
LLLL = P2 - LT.8.00-1
IF (LLLL) CALL XXXXX(0009,2)
IF (LLLL) CALL XXXXX(0010,2)
CALL XXXXX(0010,2)
DO 20 N=1,L
CALL XXXXX(0011,2)
P1 = N
P2 = N*N
R1 = RTERM
RTERM = (R1*X2-ITERM*Y2)/P2
ITERM = (R1*Y2+ITERM*X2)/P2
T1 = T1 + 0.500/P1
R0 = R0 + T1*RTERM - T2*ITERM
IM0 = IM0 + T1*ITERM + T2*RTERM
P1 = P1 + 1.000
T1 = T1 + 0.500/P1
R01 = R0 + (T1*RTERM-T2*ITERM)/P1
IM1 = IM1 + (T1*ITERM+T2*RTERM)/P1
20 CONTINUE
CALL XXXXX(0012,2)
R1 = X/R2 - 0.500*(X*R01-Y*IM1)
R2 = -Y/R2 - 0.500*(X*IM1+Y*R01)
P1 = DEAP(X)
R0 = P1*R0
IM0 = P1*IM0
R01 = P1*R01
IM1 = P1*IM1
RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE INTEGRAL
C REPRESENTATION, EON 3, EVALUATED WITH 15 POINT GAUSS-
C HERMITE QUADRATURE
30 CALL XXXXX(0013,2)
X2 = 2.000*X
Y2 = 2.000*Y
R1 = Y2*Y2
P1 = DSQRT(X2*X2+R1)
P2 = DSQRT(P1*X2)
T1 = EXSQ(1)/(2.000*P1)
R0 = T1*P2
IM0 = T1/P2
R01 = 0.000
IM1 = 0.000
DO 40 N=2,8
CALL XXXXX(0014,2)
T2 = X2 + T5Q(N)
P1 = DSQRT(T2*T2+R1)
P2 = DSQRT(P1+T2)
T1 = EXSQ(N)/P1
R0 = R0 + T1*P2
IM0 = IM0 + T1/P2
T1 = EXSQ(N)*T5Q(N)
R01 = R0 + T1*P2
IM1 = IM1 + T1/P2
40 CONTINUE
CALL XXXXX(0015,2)
T2 = -Y2*IM0
R01 = R01/R2
R2 = Y2*IM1/R2
RTERM = 1.4142135623730900*DCOS(Y)

```

KZE 500	71
KZE 510	72
KZE 520	73
KZE 530	74
*****	75
*****	76
KZE 540	77
*****	78
KZE 550	79
*****	80
KZE 560	81
KZE 570	82
KZE 580	83
KZE 590	84
KZE 600	85
KZE 610	86
KZE 620	87
KZE 630	88
KZE 640	89
KZE 650	90
KZE 660	91
KZE 670	92
KZE 680	93
*****	94
KZE 690	95
KZE 700	96
KZE 710	97
KZE 720	98
KZE 730	99
KZE 740	100
KZE 750	101
KZE 760	102
KZE 770	103
KZE 780	104
KZE 790	105
*****	106
KZE 800	107
KZE 810	108
KZE 820	109
KZE 830	110
KZE 840	111
KZE 850	112
KZE 860	113
KZE 870	114
KZE 880	115
KZE 890	116
KZE 900	117
*****	118
KZE 910	119
KZE 920	120
KZE 930	121
KZE 940	122
KZE 950	123
KZE 960	124
KZE 970	125
KZE 980	126
KZE 990	127
KZE1000	128
*****	129
KZE1010	130
KZE1020	131
KZE1030	132
KZE1040	133

```

ITEM = -1.4142135623730900*DSIN(Y)
C THE CONSTANT IN THE PREVIOUS STATEMENTS IS OF COURSE*
C SQR(2.0).
IM0 = RE0*ITERM + I2*ITERM
RE0 = RE0*ITERM - I2*ITERM
I1 = RE1*ITERM - R2*ITERM
I2 = RE1*ITERM + R2*ITERM
RE1 = I1*X + I2*Y
IM1 = -I1*Y + I2*X
RETURN
C THIS SECTION CALCULATES THE FUNCTIONS USING THE
C ASYMPTOTIC EXPANSIONS
50 CALL XXXXX(0016,2)
ITERM = 1.000
ITERM = 0.000
RE0 = 1.000
IM0 = 0.000
RE1 = 1.000
IM1 = 0.000
P1 = 6.000*R2
P2 = DSQR(R2)
L = 3.9100+8.1201/P2
R1 = 1.000
R2 = 1.000
M = -8
K = 3
00 60 N=1,L
CALL XXXXX(0017,2)
M = M + 1
K = K - M
R1 = FLOAT(K-4)*R1
R2 = FLOAT(K)*R2
I1 = FLOAT(N)*P1
I2 = ITERM
ITERM = (I2*X+ITERM*Y)/I1
ITEM = (-I2*Y+ITERM*X)/I1
RE0 = RE0 + R1*ITERM
IM0 = IM0 + R1*ITERM
RE1 = RE1 + R2*ITERM
IM1 = IM1 + R2*ITERM
60 CONTINUE
CALL XXXXX(0018,2)
I1 = DSQR(P2*X)
I2 = -Y/I1
P1 = 8.862269254527580-1/P2
C THIS CONSTANT IS SQR(P1)/2.0, WITH P1=3.14159...
ITERM = P1*DCOS(Y)
ITERM = -P1*DSIN(Y)
R1 = RE0*ITERM - IM0*ITERM
R2 = RE0*ITERM + IM0*ITERM
RE0 = I1*R1 - I2*R2
IM0 = I1*R2 + I2*R1
R1 = RE1*ITERM - IM1*ITERM
R2 = RE1*ITERM + IM1*ITERM
RE1 = I1*R1 - I2*R2
IM1 = I1*R2 + I2*R1
RETURN
99999 FORMAT (42H ARGUMENT OF THE BESSEL FUNCTIONS IS ZERO,
* 35H OR LIES IN LEFT HALF COMPLEX PLANE)
END

```

135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193